**IN THE UNITED STATES PATENT AND TRADEMARK OFFICE**

| | | |
|---|---|---|
| First Named Applicant: Campello de Souza | ) | Art Unit: 2133 |
| Serial No.: 09/899,459 | ) | Examiner: Trimmings |
| Filed: July 5, 2001 | ) | **ARC9-2001-0004US1** |
| For: SYSTEM AND METHOD FOR GENERATING LOW-DENSITY PARITY CHECK CODES USING BIT FILLING | ) | November 19, 2003<br>750 B STREET, Suite 3120<br>San Diego, CA 92101 |

---

**DECLARATION**

Commissioner of Patents and Trademarks
Washington, DC 20231

Dear Sir:

I, Dharmendra S. Modha, declare as follows:

1. I am a co-inventor of the invention claimed in the above-captioned U.S. patent application.

2. My co-inventors and I both conceived of and reduced the invention to practice at least as early as February 1, 2001, as evidenced by the enclosed document.

3. Specifically, the enclosed document entitled "Designing LDPC Codes Using Bit-Filling" and accurately dated July 31, 2000, discloses the invention of, e.g., Claim 1 for the following detailed reasons.

4. On the second page (numbered "1"), under Section 1, a computer-implemented method for generating low-density parity check (LDPC) codes is disclosed that includes constructing an m x n matrix H having m rows of check nodes and n columns of bit nodes (second paragraph of Introduction Section 1), wherein the $j^{th}$ column has weight $a_j$, each row, r, has a weight at most $b_r$ (third paragraph of Introduction),

the matrix H is representable by a Tanner graph having a girth g (second paragraph of Introduction and Abstract). The step of Claim 1 of iteratively adding an $(n+1)^{th}$ column ($U_1$) to matrix H is disclosed in the second sentence of Section 2, "The Bit-Filling Algorithm", on page number 1. The claimed feature that the size of $U_1$ is initially empty and is at most $a_{n+1}$ is stated in the third sentence. The claimed feature that $U_1$ comprises a set of i check nodes such that i is greater than or equal to 0 and i is less than $a_{n+1}$ is set forth in the next two sentences. Continuing, the claimed step of iteratively adding check nodes to $U_1$ such that each check node does not violate predetermined girth and check-degree constraints is set forth on page number 3, under equation (2). The claimed step of updating matrix H when a new column is added is disclosed in the first full paragraph of page number 2, referring to line 4 of Figure 1 on that page. The claimed step of terminating the iterations if there are no new check nodes that do not violate the girth and check-degree constraints is set forth in the last sentence of the first (partial) paragraph on page number 2.

5. The above algorithm is also set forth in Figure 1 of the enclosed document.

6. Likewise, at least the remaining independent claims are fully disclosed in the enclosed document.

7. I declare, based on first hand knowledge, that the claimed invention was actually reduced to practice at the IBM Almaden Research Facility at least as early as February 1, 2001. The claimed invention was reduced to practice by programming a computer according to the claims and producing a LDPC in the year 2000.

8. I further declare, based on first hand knowledge, that to the extent that constructive reduction to practice only is considered, my co-inventors and myself were reasonably diligent in disclosing the invention

to IBM patent attorneys and promoting the filing of a patent application in accordance with standard IBM patenting procedures at least from a time prior to February 1, 2001 until that date.

9. I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United State Code and that such willful, false statements may jeopardize the validity of the application or any patent issued thereon.

Dharmendra S. Modha

Respectfully submitted,

John L. Rogitz
Registration No. 33,549
Attorney of Record
750 B Street, Suite 3120
San Diego, CA 92101
Telephone: (619) 338-8075

JLR:jg

1033-120.05C

# DESIGNING LDPC CODES USING BIT-FILLING

Jorge Campello, Dharmendra S. Modha, Sridhar Rajagopalan

IBM Almaden Research Center
650 Harry Road, San Jose, CA 95120-6099

email: {campello,dmodha,sridhar}@almaden.ibm.com



July 31, 2000

**ABSTRACT:** Bipartite graphs of information *bits* and parity *checks* arise as Tanner graphs corresponding to low density parity check codes. Given graph parameters such as the number of checks, the maximum check-degree, the bit-degree, and the girth, we consider the problem of constructing bipartite graphs with the largest number of bits, that is, the highest rate. We propose a simple-to-implement heuristic BIT-FILLING algorithm for this problem. As a benchmark, our algorithm yields better or comparable rate codes to those published by MacKay [1].

**Keywords:** LDPC codes, Gallager codes, error correction codes, Tanner graph, parity check matrix, code design, iterative decoding

## 1. Introduction

Bipartite graphs of information *bits* and parity *checks* arise as Tanner graphs corresponding to low density parity check codes. The length of the smallest cycle in a graph is known as its *girth*. When decoding low density parity check (LDPC) codes using the sum-product decoding algorithm, the number of independent iterations of the algorithm is proportional to the girth of the Tanner graph corresponding to the code [2, Lemma C.1]. Hence, it is important to construct bipartite graphs of bits and checks that have large girth. Conversely, for fixed girth, we would like to maximize the rate of the associated code, that is, the number of checks.

A LDPC code can be described by specifying its parity check matrix $H$, which is a $m \times n$ binary matrix such that its rows correspond to checks and its columns correspond to bits. Given a parity check matrix $H$, we can define its associated Tanner graph $G(H) = (V, E)$ as a bipartite graph with $m + n$ vertices $V = \{1, 2, \ldots, m, m+1, \ldots, m+n\}$. For $1 \leq i \leq m$ and $1 \leq j \leq n$, there is an edge $(i, m+j)$ in $E$ iff $H_{i,j} = 1$. The rate of the LDPC code corresponding to $H$ is $n/(m+n)$.

In this paper, we are interested in the following design problem in constructive combinatorics. Suppose we are given positive integers $a$, $b$, $g$, and $m$; where $g$ is required to be even. We are interested in constructing a $m \times n$ parity check matrix $H$ with the largest possible rate (that is, the largest possible number of columns $n$) such that $H$ has exactly $a$ ones in each column, at most $b$ ones in each row, and $G(H)$ has girth $g$.

In this paper, we propose a simple-to-implement heuristic BIT-FILLING algorithm for the above problem. Our algorithm has a computational complexity of $O(bm^3)$ (a simpler version can be implemented using $O(bm^2)$ operations). As a benchmark, our algorithm finds better or comparable rates to some of the highest rate codes found by MacKay [1]. Our algorithm can easily be adapted to the important fixed rate case where given the number of parity checks $m$, the number of information bits $n$, the check degree $b$, and bit degree $a$, we are interested in maximizing the girth $g$.

We now briefly review various approaches to constructing LDPC codes. In his original monograph, Gallager [2, Appendix] gave an algorithm for the fixed rate construction. For various algebraic constructions, also in fixed rate setting, see [3, 4, 5, 6]. For various recent designs of graphs with girth 6 using mainly random constructions, see, for example, [7, 8]. For designs of irregular graphs usind random constructions and linear programming, see [9].

## 2. The BIT-FILLING Algorithm

### 2.1. The Idea

Suppose we have already constructed a matrix $H$ with $n$ columns, $n \geq 0$, that satisfies all the constraints. We now show how to add $(n+1)$-th column to $H$. We think of the new column

```
1:  set n = 0, A = |m|, and U_1 = φ
2:  for c ∈ |m|, set weight(c) = 0 and N_c = φ
3:  do {
4:      ∀c ∈ U_1, set H_{c,n} = 1
5:      set i = 0, U_1 = φ, and U = φ
6:      do {
7:          compute F_0 = A \ U
8:          if (F_0 ≠ φ) {
9:              choose c* from F_0 according to some heuristic
10:             ∀c ∈ U_1, update N_c = N_c ∪ {c} and update N_{c*} = N_{c*} ∪ U_1
11:             update U_1 = U_1 ∪ {c*}, U = U ∪ V_{(g/2)-1}(c*), degree(c*), and A
11:         }
12:         i = i + 1
13:     } while ((i < a) and (F_0 ≠ φ))
14:     n = n + 1
15: } while ((i = a) and (F_0 ≠ φ))
```

Figure 1: The BIT-FILLING Algorithm.

to be added as a set $U_1$ which has size at most $a$ and is initially empty. The set $U_1$ is a set of checks and, hence, is a subset of $|m| \equiv \{1, 2, \cdots, m\}$. Further suppose we have already added $i$ checks, $0 \leq i < a$, to $U_1$. The following procedure attempts to add $(i+1)$-th check to $U_1$. It may fail to add $(i+1)$-th check, and, in that case, the whole procedure stops.

In Figure 1, we present the algorithm based on the outline sketched above. Each iteration of the outer "do $\cdots$ while" loop in lines 3–15 attempts to add a column to the parity check matrix. If an iteration succeeds, then, in the beginning of the next iteration, line 4 adds updates the matrix $H$. Each iteration of the inner "do $\cdots$ while" loop in lines 6–13 attempts to add a check to the set $U_1$.

To describe the procedure, it helps to think of the Tanner graph $G \equiv G(H)$ associated with the matrix $H$. The bipartite graph $G$ has $m$ check nodes and $n$ bit nodes. The process of adding $(n+1)$-th column of $H$ is like adding $(n+1)$-th bit node to $G$. In this context, $U_1$ can be thought of as the set of checks that have been already connected to the $(n+1)$-th bit node that is being added. The process of adding a check $c^*$ to $U_1$ is like adding an edge from the $(n+1)$-th bit to $c^*$-th check. This new edge must not create any cycles of length $(g-2)$ or smaller; we now describe a test to enforce this constraint.

For a check $1 \leq c \leq m$, let $\mathcal{N}_c$ denote the set of all checks that share a bit with it. In other words, $\mathcal{N}_c$ is the set of all check nodes that are exactly two distinct edges away from $c$. For

$j \geq 2$, define

$$U_j = \bigcup_{c \in U_{j-1}} \mathcal{N}_c. \tag{1}$$

Intuitively, there is a path of length 2 from every check in $U_2$ to some check in $U_1$. Now adding a check $c^*$ to $U_1$ will create a path of length 2 from $c^*$ to every check in $U_1$. So, if $c^*$ is in $U_2$, then we are guaranteed a 4-cycle. Hence, to avoid 4-cycles, we should avoid the checks in $U_2$. Continuing in this fashion, there is a path of length 2 from from every check in $U_j$ to some check in $U_{j-1}$, and, hence, there is a path of length at most $2j-2$ from from every check in $U_j$ to some check in $U_1$. Thus, adding a check in $U_j$ to $U_1$ will create a cycle of length $2j$ or smaller. Hence, to satisfy the girth constraint, we should avoid adding checks in the set

$$U = \bigcup_{1 \leq j \leq (g/2)-1} U_j \tag{2}$$

to $U_1$.

Let

$$A = \{c \in |m| : \text{degree} < b\}$$

denote set of checks that are connected to fewer than $b$ bits. Then, the set of feasible checks that can be added to $U_1$ without violating the girth or the check-degree constraint is

$$F_0 = A \setminus U.$$

The procedure terminates if $F_0$ is empty. Before we take up the important issue of how to select a check from $F_0$, we briefly discuss an efficient implementation of (2).

Observe that, as checks are added to $U_1$, it is computationally more efficient to incrementally update $U$ in line 10 than recomputing it afresh using (1)–(2). Specifically, if we have added a check $c$ to $U_1$, then we set $U_1(c) = \{c\}$ and, for $2 \leq j \leq (g/2)-1$, compute

$$U_j(c) = \bigcup_{c' \in U_{j-1}(c)} \mathcal{N}_{c'} \tag{3}$$

and compute

$$V_{(g/2)-1}(c) = \bigcup_{1 \leq j \leq (g/2)-1} U_j(c). \tag{4}$$

Finally, update $U$ using $U = U \cup V_{(g/2)-1}(c)$.

## 2.2. Main Heuristic

The most crucial step of the BIT-FILLING algorithm is the choice of the check $c \in F_0$ in line 9 of Figure 1. While any choice of $c^*$ from $F_0$ is valid, judicious selection of $c^*$ is crucial to achieving high-rate codes. Technically, using back-tracking and recursion, we can try every possible choice, and then "conditioned" on that choice try every future choice, etc., thus searching through an extremely large tree of possibilities. Of course, we often wish we could live life in this way! Unfortunately, while optimal, such exhaustive search is computationally infeasible. Also, a large number of choices actually lead to isomorphic graphs and are thus equivalent from our perspective. The role of heuristics is to pick one seemingly effective, and yet computationally feasible, path through this gargantuan tree of possibilities.

We let our choice of $c^*$ in line 9 be guided by the simple principle of keeping the Tanner graph $G$ as *homogeneous* as possible, that is, from the set of all feasible checks pick the check that is least used so far. This choice amounts to keeping all parts of the graph equally dense. Also, from the perspective of the $(n+1)$-th bit that is being added, this choice connects it to the check that is least used, and, hence, provides better error protection. We now make our choice precise.

As a first try, we look at the following subset of $F_0$

$$F_1 = \{c_1 \in F_0 : \forall c_2 \in F_0, \text{degree}(c_1) \leq \text{degree}(c_2)\} \tag{5}$$

namely, the set of smallest degree checks in $F_0$. As a FIRST-ORDER heuristic, we may simply choose $c^*$ to be any element of $F_1$. We shall refer to these heuristic as "1-h" meaning first-order homogeneity. We will show in Section 3 that this first-order heuristic already yields quite competitive codes.

Typically, while $F_1$ is smaller than $F_0$, it does not uniquely determine a check. To further narrow available choices, we appeal once more to homogeneity. The idea is to look at the degrees of checks that are two edges away from the checks in $F_1$. We write

$$F_2 = \left\{ c_1 \in F_1 : \forall c_2 \in F_1, \sum_{c' \in V_2(c_1)} \text{degree}(c') \leq \sum_{c'' \in V_2(c_2)} \text{degree}(c'') \right\},$$

where $V_2(\cdot)$ is as in (4). Now, we may try selecting a check from $F_2$. However, once again, it is possible that $F_2$ has more than one element. Thus, to further narrow the set of choices, we may look at the degrees of checks that are four edges away from the checks in $F_1$, and can continue in this fashion. The basic idea can now be described algorithmically, see Figure 2. The idea is to progressively look at larger and larger neighborhoods of a feasible checks in order to distinguish them; we refer to this heuristic as "c-h" for complete-homogeneity. The "while ... endwhile" loop terminates when the set of choices reduces to a set of cardinality one or till the set of choices does not further reduce in cardinality. The set $E_j$ in line 9H is a set of

all checks in $F_j$ such that their neighborhood can be further enlarged. The loop terminates if $E_j \neq F_j$, since, in this case, the checks in $E_j$ are guaranteed to be less homogeneous than those in $F_j \setminus E_j$. Furthermore, since, using homogeneity as our guide, we cannot further distinguish between the elements of $F_j \setminus E_j$, we simply make some choice from this set. Specifically, we select the lexicographically smallest check from this set.

```
9A:   set j = 0, E_0 = F_0
9B:   ∀c ∈ F_0, set U_1(c) = {c}
9C:   while ((|F_j| > 1) and (E_j = F_j)) {
9D:       j = j + 1
9E:       ∀c ∈ F_{j-1}, compute W_j(c) = Σ_{c'∈V_j(c)} degree(c')
9F:       compute F_j = {c_1 ∈ F_{j-1} : ∀c_2 ∈ F_{j-1}, W_j(c_1) ≤ W_j(c_2)}
9G:       ∀c ∈ F_j, compute V_{j+1}(c) using (4)
9H:       compute E_j = {c ∈ F_j : V_{j+1}(c) ≠ V_j(c)}
9I:   } endwhile
9J:   if (|F_j| = 1)
9K:       select c* from F_j
9L:   else
9M:       select c* from F_j \ E_j
9N:   endif
```

Figure 2: The "c-h" heuristic for selecting the check $c^*$ from $F_0$. This heuristic is meant to replace line 9 in Figure 1.

## 3. Results and Discussion

**Remark 3.1 (fixed girth, high rate)** MacKay has collected a large number of LDPC codes at [1]. Amongst the codes listed there, he singles out 9 codes as "the following numbers give the highest rate codes I have made with column weights 3 and 4." In Table 1, we compare these codes to those generated by our algorithm. It is evident that in 7 out of 9 cases our algorithm yields higher rate codes, and in 2 cases, while not better, it yields virtually identical rates.

**Remark 3.2 (fixed rate, high girth)** So far we focussed on maximizing the rate given constraints on the number of parity checks, check degrees, bit degrees, and the girth. Another interesting problem, first considered in Gallager [2], is given the number of parity check bits $m$, the check degree $b$, and bit degree $a$, maximize the girth $g$. For various algebraic constructions of fixed rate, high girth graphs, see [3, 4].

| $a$ | $m$ | $n$ | | |
|---|---|---|---|---|
| | | | BIT-FILLING | |
| | | MacKay | 1-h | c-h |
| 3 | 100 | 900 | 1229 | **1339** |
| 3 | 111 | 999 | 1515 | **1636** |
| 4 | 222 | 1998 | 2628 | **2752** |
| 4 | 300 | 4096 | 4807 | **5499** |
| 4 | 444 | 3584 | 10839 | **11088** |
| 3 | 60 | **492** | 437 | 485 |
| 3 | 62 | 494 | 464 | **483** |
| 3 | 90 | 998 | 970 | **1087** |
| 4 | 282 | 4376 | 4293 | **4821** |

Table 1: The "MacKay" column refers to 9 codes in [1] while the "1-h" and "c-h" columns refer to selecting $c^*$ in line 9 of Figure 1, respectively, using the first-order heuristic in (5) and using the complete-homogeneity argument in Figure 2. Also, for all codes, we required $g = 6$ and $b$ was unrestricted.

In this case, the number of information bits is $n = (mb)/a$. Our algorithm can be adapted to this case as follows. We simply apply the algorithm with given parameters $m$, $a$, $b$, and some girth $g$. If the algorithm finds exactly (resp. less than) $n$ bits, then the girth $g$ is achievable (resp. not achievable) with our algorithm. By trying a few girth values (formally, doing a binary search on girth values), one can quickly determine the largest girth that is attainable using our algorithm.

Once again, we use the encyclopedia of codes compiled by MacKay [1] as our benchmark. All the Gallager codes listed in [1] have girth 6, that is, avoid 4 cycles. For all of these codes, our algorithm matches the girth. More interestingly, for some of these codes, our algorithm is able to improve the girth parameter. For example, for two rate 1/2 codes with $a = b = 3$ and $m = 48$ and 102, our algorithm is able to improve the girth from 6 to 8.

**Remark 3.3 (computational hardness)** To analyze the computational complexity of the problem of designing high rate codes satisfying given parameters, we consider the following decision problem.

GIRTH&BI-DEGREE
*Instance:* $(a, b, g, m, n)$ where $a$, $b$, $g$, $m$, and $n$ are positive integers and $g$ is even.
*Question:* Does there exist a $m \times n$ binary matrix $H$ such that $H$ has exactly $a$ ones in each

column, at most $b$ ones in each row, and $G(H)$ has girth $g$.

We take size of the instance to be $a+b+g+m+n$, and not $\log(abgmn)$. Now, given a $m \times n$ binary matrix $H$, the constraints on the weights of the rows and the columns are easy to check. Furthermore, using roughly $O((n+m)^2)$ operations it is easy to determine the girth of $G(H)$. Hence, the problem GIRTH&BI-DEGREE is clearly in **NP**. However, the input consists of simply five integers, and does not contain any additional structure. Hence, assuming that $\mathbf{P} \neq \mathbf{NP}$, it follows from Berman [10] that this problem cannot be **NP**-hard.

**Remark 3.4 (computational complexity)** We now roughly analyze the computational complexity of the algorithm in Figure 1. Line 7 implements a set difference, and can be implemented using $O(m)$ operations. Line 9 finds the check $c^*$ using the algorithm in Figure 2, and can be implemented, in the worst case, using $O(m^2)$ operations. Updating $U_1$ and $\mathcal{N}_c$ on line 10 can be both be implemented using $O(a)$ operations, updating $A$ can be implemented using $O(m)$ operations, and, using (3)–(4), updating $U$ can be implemented using $O((ab)^{g-1}) = O(m)$ operations. Hence, each iteration of the inner "do ··· while" loop in lines 6-14 can be implemented in $O(m^2)$ operations. Since, the loop is iterated at most $a$ times, the entire loop in lines 6-14 can be implemented in $O(am^2)$ operations. Finally, the outer "do ··· while" loop in lines 3-16 is iterated at most $(mb)/a$ times, and, hence, the entire algorithm requires $O(bm^3)$ operations. Observe that if, in line 9, we use the first-order heuristic algorithm in (5) instead of the entire algorithm in Figure 2, the entire resulting algorithm requires only $O(bm^2)$ operations.

## Acknowledgments

We are grateful to Brian Marcus for a number of valuable discussions.

## References


[1] D. J. C. MacKay, "Encyclopedia of sparse graph codes," 1999. http://wol.ra.phy.cam.ac.uk/mackay/codes/data.html.

[2] R. G. Gallager, *Low-density parity-check codes*. MIT Press, Cambridge, MA, 1963.

[3] G. A. Margulis, "Explicit group-theoretical construction of combinatorial schemes and their application to the design of expanders and concentrators," *Problems of Inform. Transmission*, vol. 24, no. 1, pp. 39–46, 1988.

[4] A. Lubotzky, R. Phillips, and P. Sarnak, "Explicit expanders and the ramanujan conjectures," *Combinatorica*, vol. 8, no. 3, pp. 261–277, 1988.

[5] F. Lazebnik and V. A. Ustimenko, "New examples of graphs without small cycles and of large size," *Europ. J. Combinatorics*, vol. 14, pp. 445–460, 1993.

[6] Z. Füredi, F. Lazebnik, A. Seress, V. A. Ustimenko, and A. J. Woldar, "Graphs of prescribed girth and bi-degree," *J. Combinatorial Theory, Series B*, vol. 64, pp. 228–239, 1995.

[7] D. J. C. MacKay, S. T. Wilson, and M. C. Davey, "Comparison of constructions of irregular Gallager codes," *IEEE Trans. Communications*, vol. 10, pp. 1449–1454, October 1999.

[8] D. J. C. MacKay and M. C. Davey, "Evaluation of Gallager codes for short block length and high rate applications," in *Proceedings of the IMA Workshop on Codes, Systems, and Graphical Models*, 1999.

[9] M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. Spielman, "Analysis of low-density codes and improved designs using irregular graphs," in *Proceedings of the 30th annual ACM Symposium on Theory of Computing*, pp. 249–258, 1998.

[10] P. Berman, "Relationship between density and deterministic complexity of np-complete languages," in *Fifth Intl. Colloq. on Automata, Languages, and Programming, Lecture Notes in Computer Science*, vol. 62, pp. 63–71, 1978.